

POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione

Corso di Laurea Magistrale in
Computer and Science Engineering



Digital Healthcare, a web application
for Doctors and Volunteers

Advisor: Prof. Luciano BARESI

Co-Advisor: Ing. Carlo GERI (Medici Volontari Italiani - ETS)

Thesis by:

Davide LAFFI

Matr. 941535

Academic Year 2020 - 2021

Abstract

The purpose of this thesis is the design and the implementation of a web application to support family medicine physician. The idea is to create an innovative system to manage citizens, in this particular case of Lombardia region, that can bind together both the medical side, inserted by doctors and the general social information side, inserted by a third party, like municipal volunteer. This would allow to have a real improvement in the efficiency of filling the citizens data and a greater clarity in showing information too. So, the application is designed both for family medicine physicians and for the above municipal volunteers. In the first phase a doctor should insert all medical data of a patient according to the guidelines of the HER/PS, (FSE/PSS in Italian) [1], that is the Patient Summary of the Electronic Health Report, and then a volunteer has to complete citizen's data with general information like, ID card, ICE numbers. In this way there would be an overall profile of the citizen. The other scope of the application is about first medical aid. In case either of accident or urgency medical situation, an application made by a colleague of mine will use data registered by my application to generate, in an intuitive way like with a QR code, a PDF file with the citizen profile. That will be composed by all the most important data which can be used in an emergency, for example blood group, relevant diseases and emergency contacts which can save his life.

Sommario

L'obiettivo del progetto di tesi è la progettazione e lo sviluppo di un'applicazione web a supporto dei medici di base. L'idea è quella di creare un sistema innovativo che gestisca i cittadini, in questo caso particolare della regione Lombardia, e che possa unire insieme sia la parte medica, inserita dai medici, sia quella più generale, inserita da un terzo ente, per esempio un volontario comunale. Questo porterebbe a un grande miglioramento nell'efficienza del riempimento dei dati dei cittadini e una maggiore chiarezza nella visualizzazione delle informazioni. Quindi l'applicazione è progettata sia per i medici di base, sia per i volontari comunali. Nella prima fase il medico inserisce tutti i dati relativi alla parte medica del cittadino, seguendo le linee guida del Fascicolo Sanitario Elettronico, e in secondo luogo, un volontario deve completare il profilo del cittadino con i dati più generali, come carta d'identità, numeri ICE. In questo modo si creerà un profilo completo del cittadino. Il secondo scopo dell'applicazione riguarda l'intervento di primo soccorso. In caso di incidente o di urgenza medica, un'applicazione progettata e sviluppata da un mio collega userà i dati registrati dalla mia applicazione per generare, in modo intuitivo per esempio con un codice QR, un file PDF con il profilo del cittadino. Questo file sarà composto dai dati più importanti che hanno rilevanza in caso di emergenza come per esempio il gruppo sanguigno, malattie rilevanti e contatti di emergenza che possono salvargli la vita.

Contents

Contents	7
List of Figures	9
1 Introduction	11
1.1 Overview	11
1.2 State of the art	12
1.3 Goal of the project	15
1.3.1 Limits and issues of the State of the art	15
1.3.2 Improvements and goal of the project	16
2 Requirement Analysis	19
2.1 User Requirements	19
2.2 Use Cases	21
2.3 Sequence Diagram	22
3 Architecture Design	29
3.1 Architecture Overview	29
3.2 Angular Framework	30
3.2.1 Typescript	31
3.3 Server side: Firebase	32
3.3.1 Firebase Authentication	33
3.3.2 Firebase Storage	34
3.3.3 Firestore Database	35
4 Implementation	39
4.1 Packages diagram	39
4.2 Model	40
4.3 Main classes	43
4.3.1 Login and registration	43
4.3.2 Home	46

4.3.3	Add new Patient	48
4.3.4	Patient profile	50
4.4	Services	51
4.4.1	Database	51
4.4.2	Storage	53
4.4.3	Guard	54
4.4.4	Pipes	54
5	Functionalities and demonstration	57
5.1	Login and registration	57
5.2	Home	58
5.3	Add new citizen	59
5.3.1	Add new patient by doctor	59
5.3.2	Add new citizen by volunteer	60
5.4	Citizen Profile	62
5.4.1	Patient's profile	62
5.4.2	Citizen's profile	63
5.5	Covid19 emergency	64
5.6	User's profile	66
6	Conclusions and future improvements	67
	Bibliography	69

List of Figures

1.1	EHR guidelines for the medical profile	13
1.2	Carta di identità salvavita	13
1.3	CIS creation	14
1.4	CIS example	15
1.5	Architecture Design	17
2.1	Data entry	20
2.2	Use Cases Doctor	21
2.3	Use Cases Volunteer	22
2.4	add new patient sequence diagram	23
2.5	Patient Profile visualization sequence diagram	23
2.6	Covid-19 emergency sequence diagram	24
2.7	Personal profile sequence diagram	24
2.8	Add Citizen sequence diagram	26
2.9	Covid-19 emergency volunteer sequence diagram	26
3.1	General architecture	30
3.2	New Patient Component	32
3.3	Authentication	34
3.4	Storage	34
3.5	General architecture	35
3.6	Citizens collection	36
3.7	covid19 collection	36
3.8	data collection	37
4.1	Packages diagram	39
4.2	Model diagram	40
4.3	Packages list	43
4.4	Login and registration package	43
5.1	Login page	57
5.2	Registration phase	58

5.3	Home page	58
5.4	Add new patient	59
5.5	New patient form	60
5.6	Citizens' list	61
5.7	Citizen's profile	61
5.8	Patients' list	62
5.9	Patient's profile	62
5.10	Citizen's profile	63
5.11	Covid19 section	64
5.12	Add new report	65
5.13	Covid19's section volunteer	65
5.14	User's profile	66

Chapter 1

Introduction

1.1 Overview

Digital Healthcare is a system made in collaboration with a colleague of mine, Andrea Calici, which has the aim to create a link between citizens data. This connection between data has borned because the thesis' goal is to create a system which could help medical first aid. When there is either an accident or a medical urgency, time is the most important variable. Normally, in this type of situations, rescuers have to lose a lot of time in searching information about the person involved in the accident. As a matter of fact, it is the same in hospitals, when people arrive to the emergency room, doctors have to identify the patient and collect all his data. The idea is to create a QR code which encapsulates fundamental information of a citizen like blood group, allergies, and relevant diseases, in order for the doctors to be able to intervene as soon as possible, having an overview of the most relevant information. At the state of the art, doctors have to collect data which can have missing fields, or they can be old or without an historical, but also they can come from different sources which complicates the situation. Nowadays each family medicine physician has his own software to register patients' health data, while municipal volunteers have another software to register more general data like ID card and so on. Moreover, even between doctors there are different software to use, which guarantee a heterogeneous quantity of data. The idea is to create a system which completely deletes differences and compatibility problems between data. So, the goal is to design first of all, a robust database in which all family medicine physician and municipal volunteers can write and read from, and, in addition, to create a mobile application, based on this database, which is focused on medical first aid. After discussing with Andrea Calici, we agreed that I would devolop the web application, that can

be used also as a desktop application in the future, and he would program a mobile application focused on the medical first aid, while the database would be designed in common. This idea has borned in collaboration with *Medici Volontari Italiani* [3] that is an association created at the end of 90's and that became an ONLUS in 2005. The project was made possible with the help and the supervision of Carlo Geri, who is a member of the association that has helped us during the user requirements phase and checked the work during the whole project. Talking with the customer some other features and requirements have came on and it was introduced the possibility to create an historical of all the citizens to keep monitored the health profile. Moreover, the project has been started during the pandemic situation due to Covid-19 and so, being a medical application, it was introduced a Covid-19 emergency section to handle tests, vaccinations, and Green Pass, which is the European certification that certify the protection from the virus. To sum up, the system is composed by a web application used by doctors, in particular by family medicine physicians, which provides a software to insert and register patients and their data related to the medical part. The same web application has a second section addressed to municipal volunteer to complete the data of patients with the more general part of information like where do they live, ID card, picture, and so on. Then there is a database which is in common to the second part of the project, that is used to keep the data and to create a linkage between the two applications. To conclude there is a mobile application which allows to generate the QR code and to print all the reports needed for a citizen.

1.2 State of the art

As it was said before, at the current situation handling data is complicated. There is not a unique software which allows family medicine physicians to insert data. So, each doctor registers data of patients in a different way, with different format and increases the incompatibility of information. Moreover, municipal volunteers add the general data of citizens to a completely different platform, so that linking data becomes impossible.

Versione 31 marzo 2014

PROFilo SANITARIO SINTETICO						
Sezione	Elemento	Contenuto informativo	Descrizione	Obbligatorietà	Testo libero / Codificato	Fonte di riferimento
SEZIONE INTESTAZIONE						
Intestazione	Dati identificativi del paziente	Cognome assistito	Cognome dell'assistito	Obbligatorio	Testo libero	Anagrafe assistiti
Intestazione	Dati identificativi del paziente	Nome assistito	Nome completo dell'assistito (come risulta in anagrafe)	Obbligatorio	Testo libero	Anagrafe assistiti
Intestazione	Dati identificativi del paziente	Codice fiscale assistito	Codice fiscale dell'assistito	Obbligatorio	Codificato	Agenzia Entrate
Intestazione	Dati identificativi del paziente	Sesso assistito	Genere dell'assistito	Obbligatorio	Codificato	[M/F]
Intestazione	Dati identificativi del paziente	Data di nascita assistito	Data di nascita dell'assistito	Obbligatorio	Codificato	[GG/MM/AAAA]
Intestazione	Dati identificativi del paziente	Comune di nascita assistito	Comune di nascita dell'assistito	Obbligatorio	Codificato	ISTAT
Intestazione	Dati identificativi del paziente	Indirizzo di domicilio assistito	Indirizzo del domicilio dell'assistito	Obbligatorio	Testo libero	
Intestazione	Dati identificativi del paziente	CAP domicilio assistito	CAP del domicilio dell'assistito	Obbligatorio	Codificato	CAP Poste Italiane
Intestazione	Dati identificativi del paziente	Comune domicilio assistito	Comune del domicilio dell'assistito	Obbligatorio	Codificato	ISTAT
Intestazione	Dati identificativi del paziente	Recupero telefonico assistito	Recupero telefonico dell'assistito (fisso e/o mobile)	Facoltativo	Testo libero	MMG/PLS
Intestazione	Dati identificativi del paziente	e-mail assistito	Indirizzo e-mail dell'assistito	Facoltativo	Testo libero	MMG/PLS
Intestazione	Dati identificativi del paziente	PEC assistito	Indirizzo PEC dell'assistito	Facoltativo	Testo libero	MMG/PLS
Intestazione	Dati identificativi del paziente	Esenzioni assistite per patologia	Eventuali codici di esenzione del pagamento del ticket dell'assistito	Obbligatorio se applicabile	Codificato	Codifica Nazionale

Figure 1.1: EHR guidelines for the medical profile

In *Figure 1.1* is reported an extract of the guidelines from the DPCM "FSE" 31-03-2014 [2] which are effective from 31st of March 2014 to now. There are 6 pages and each row represent a field to fill in for a patient, in total there are 46 attributes. For each field there is the group, a small description, if the field is either required or not and a possible codification. This document was sent by the customer, it is the document that each family medicine physician follows in Italy, and it provides the rules to follow. The problem is that the software to insert this data is different for each hospital or office, and not all fields have a codification. This creates confusion and heterogeneity of data. In 2011, thanks to the IBM Foundation Project: "Celebration of Service", Medici Volontari Italiani [3] was able to develop an application that allows to digitalize medical data to make them easily accessible by rescuers in case of emergency. This was achieved creating the CIS which means Carta di identità Salvavita [1] that can be seen in Figure 1.2.



Figure 1.2: Carta di identità salvavita

CIS is a paper where are reported personal data of a citizen together with some relevant diseases and allergies, but also with two numbers which can be called in case of emergency. The application of Medici Volontari Italiani [1]

allows to read the QR code in the right side of the paper, in this way rescuers can have a first overview of the citizen who needs help. As it can be read on the website of Medici Volontari Italiani [1], the application exists only as a web application for medical staff and cannot be downloaded by other people. In addition to the CIS, there is the Busta Rossa [4] which contains personal information and clinic information, and the Braccialetto Salvavita [4] which is a bracelet that indicates that the person has the CIS to consult. Both are useful facilities that can help in case of emergency or clinic situations.



il telefonino, il tuo salvavita ICE+

LETTO [Informativa Privacy](#)

INFORMAZIONI PERSONALI

Nome e Cognome:

Data di nascita:
(formato gg/mm/aaaa)

Email*:

CARICA FOTO

Si consiglia di caricare una foto tessera recente e possibilmente di forma quadrata. I formati accettati sono: jpg, png e gif.

Nessun file selezionato

CONTATTI DI EMERGENZA - ICE

Nome e Cognome ICE 1:

Telefono 1:

Nome e Cognome ICE 2:

Telefono 2:

INFORMAZIONI MEDICHE SALVAVITA

Gruppo sanguigno*: 0 A B AB non specificato

Fattore Rh*: Positivo Negativo non specificato

Patologie:
(vedi S)

- Allergie SAI (995-3)
- Cardiopatia ischemica (410* 414*)
- IMA pregresso (412)
- Aneurismi: aorta (441*)
- Aneurismi: altre sedi (442*)

Figure 1.3: CIS creation

On the website of Medici Volontari Italiani [3] there is a demo application that allows to try to create your personal CIS. In *Figure 1.3* is reported the form to fill in to generate the badge where the citizen is asked to provide his personal information, the two ICE numbers, and his blood group together with relevant diseases.

il telefonino,
il tuo
salvavita ICE+

INFORMAZIONI PERSONALI
 Nome e Cognome Davide Laffi
 Data di nascita 20/03/1997
 Email davide.laffi@gmail.com

CONTATTI D'EMERGENZA ICE
 Nome e Cognome ICE 1 Mamma Silvia
 Telefono 1 395325435
 Nome e Cognome ICE 2 Papà Gilberto
 Telefono 2 34523432535

INFORMAZIONI MEDICHE SALVAVITA
 Gruppo sanguigno A
 Fattore Rh neg
 Patologie
 • Aritmia cardiaca (427*)
 • Diabete (250*)

Allergie ed intolleranze note Nessuna
 Informazioni importanti non specificato

Informazioni mediche Salvavita condivise con SIMEU, Società Italiana di Medicina d'Emergenza e d'Urgenza, a novembre 2015

Figure 1.4: CIS example

1.3 Goal of the project

1.3.1 Limits and issues of the State of the art

The limits and the issues of the actual configuration have been already analysed inside the overview of this document, but with the explanation of the state of the art they are now completely clear. The first relevant problem is the heterogeneity of data and the impossibility to create a system which encapsulates all citizens information. The second issue is a direct consequence of the first one, because without a unique and stated way to collect data, it is impossible to create a system which reads this data and that uses them. The goal is to create an application to speed up the first medical aid, so, in case of emergency, that shows to rescuers the most relevant data which can save patients life. But this cannot be done because medical data are collected by family medicine physicians in different ways, not uniform, while general data of citizens, like ID card, ICE numbers are collected by Medici Volontari Italiani [3] using another different form which the citizen himself has to fill in. The problem is that each entity has his own method to retrieve citizen data and even if the fields and the attributes are the same among the different software and platforms, they are not compatible because of the typology of the data. A date can be saved as date, as a number or even as a string, this causes confusion and impossibility to link data. Another relevant problem is that databases are different. Each doctor saves information in the database

of the system he is using, which is different respect to others. The same for volunteers which use a completely different way of storing data because they are a different entity. The problems illustrated before are not issues only in order to implement an efficient method that helps first medical aid, but they are also a real problem in data integrity. Having different ways of storing data and different types of them lead to problems in data, like duplication, missing data or even the loss of them. When you have to transfer data from a system to another, the possibility of losing data is very high in particular if the two systems are decoupled.

1.3.2 Improvements and goal of the project

The system that is created in this project wants to solve all the issues reported above. The solution is creating a new system to collect data, which is completely linked to the one which provides data when they are asked. Me and Andrea have found the first solution in having a shared database between entities. In this way, family medicine physicians insert information in a database which is read by the application of the medical first aid. Then the software of the web application would guarantee to have the same format of data for all doctors who are using it and it would register data on the database we are talking about. At the end of the process there is the mobile application which uses the information kept in the database and generates quickly and in a clear way data needed for medical first aid. This is the starting point that was agreed with the customer of Medici Volontari Italiani[3]. Clearly, thinking about the project, different improvements where both asked from the customer and proposed by me. To sum up, there is a web application which is dedicated to family medicine physicians and it is used to collect patients' data. The goal is to create a system that is used by all the doctors in Italy to fill in information. The web application generates a form following the guidelines from the DPCM "FSE" 31-03-2014 [2] for each patient in order to have precise rules which guarantee uniform data. Moreover, the system is extended to covid19 emergency, so that even potential information related to the pandemic situation are clearly organised. Then there is a mobile application focused on medical first aid which wants to provide a fast and simple support in emergency situation. The linking point between the applications is the database. In this way data are stored in the same place, allowing to retrieve them in the fastest possible way.

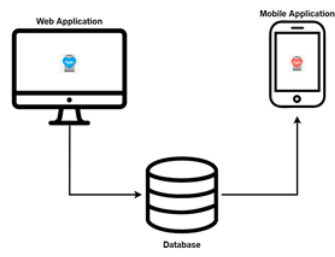


Figure 1.5: Architecture Design

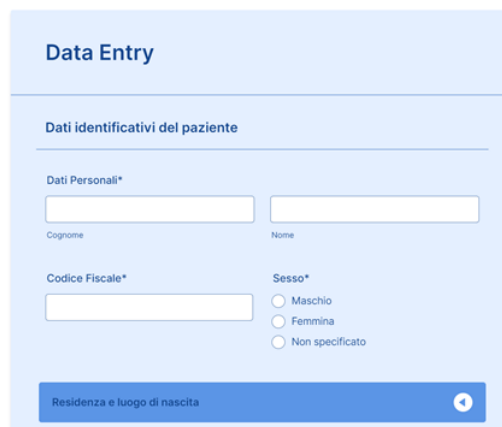
Chapter 2

Requirement Analysis

2.1 User Requirements

The work started with a conference call with the customer which explained the requirements and what he expected from the application to do. The basic requirement is an application which could be either a web application or a desktop one, which allows family medicine physicians to register medical data of a new patient inside a database. Then, in the same application, a municipal volunteer should have the possibility to access and complete the citizen's profile with more general information, like ID card number and emergency contacts. Talking about medical side, the doctor should be able to register to the application and log in, then he has a personal section in which he can add a new patient from scratch. Because of the pandemic situation due to COVID-19, according to the customer it was decided to add a dedicated section where the doctor can manage, for each of his patients, covid tests reports and vaccinations. Then there is another section in which the doctor can manage his personal profile. He can either change some of his personal data, for example PEC or phone number or other information like working days. Another important request was the possibility to keep track of relevant changes about patient health that was solved introducing an historical record which can register the past changes related to a patient's profile. At the end, the doctor should have a list of patients that can access to choose one of them in order to visualize his data. The application should give the possibility of changing data visualization, depending on the date on which information where inserted. Moving towards volunteer's part, the user should be able to register and log in to the application, and then manage citizens' profile. He can complete general information data choosing the citizen from the list of patients the doctor has added. The application should separate people who

already have a volunteer assigned, and so should have the profile complete, and people who are not assigned yet. When a person is selected the volunteer has to fill in the general information. As in doctor case, volunteers should have a personal area in which they can manage his personal profile changing common information like PEC and phone number and other one like days of works and availabilities. Even in the case of volunteer, there is a COVID-19 section, in which the volunteer can see all the reports about covid tests and vaccinations and he can also print the Green Pass in case that the citizen meets the requirements to have it. All these requirements were reached with conference calls and mail flows. An important step was the creation of the form which allows to enter data for a new patient. The customer provided a schema to follow called “Profilo Sanitario Sintetico” which is a list of fields that a family medicine physician must fill in to register a new patient. Then there was another form, created by “Medici Volontari Italiani” which asks for patient’s general social information. So, I tried to link the two schemas filtering only relevant information and required ones. During this phase, it was used a web application called JotForm [5] which allows to create mock-up forms without writing any line of code. This was very helpful because simplified the communication with the costumer by creating different mock-up versions of the data entry form. The solution that was applied is to follow accurately the schema proposed by “Profilo Sanitario Sintetico”, which is the effective schema that family medicine physician follows in Lombardia region, for the part of doctors. While, for the part of volunteer, the form provided by “Medici Volontari Italiani”[3] was used, changing some fields and avoiding repetitions.



The image shows a web form titled "Data Entry" with a light blue background. The form is divided into sections. The first section is "Dati identificativi del paziente". Below this, there is a sub-section "Dati Personali*" containing two input fields: "Cognome" and "Nome". Below these, there is a "Codice Fiscale*" input field and a "Sesso*" section with three radio button options: "Maschio", "Femmina", and "Non specificato". At the bottom of the form, there is a blue bar with the text "Residenza e luogo di nascita" and a small icon of a house.

Figure 2.1: Data entry

2.2 Use Cases

Use case diagrams are sketches that tries to represent the user flow inside the application. There are actors and states. An actor is a type of user inside the application, in this particular situation we will have an actor characterized by a “New User” and other two actors representing the doctor and the municipal volunteer. States have the goal to encapsulate all the possible situation which a user can face. For example, in Digital Healthcare a possible flow of states and actions can be a doctor who goes to *Authentication* section and then to the *Homepage* and chooses to *add a new patient*.

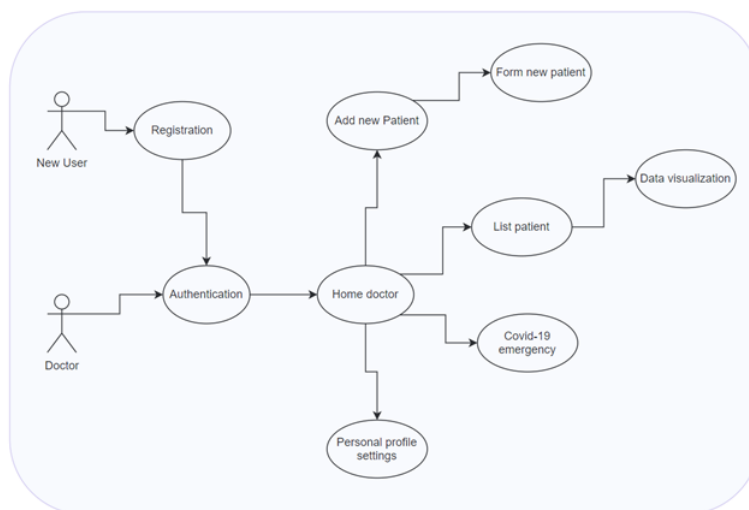


Figure 2.2: Use Cases Doctor

This is the use case diagram that sketches the user flow of a doctor. First of all, if the user is trying to enter in the web application for the first time, he will face a registration form which collects some general information about the doctor. Then the doctor fills in the login form and he can access to his personal home page, that’s in the diagram at *Figure 2.2* is called *Home doctor*. From the homepage the doctor can now choose among the functionalities of Digital Healthcare. *Add new Patient* guides the family medicine physician through an intuitive form which registers a new patient, while *List Patient* provides the list of all the patients registered in the database. The other two sections are *Covid-19 emergency* which allows to manage covid tests reports and vaccination, and *personal profile settings*.

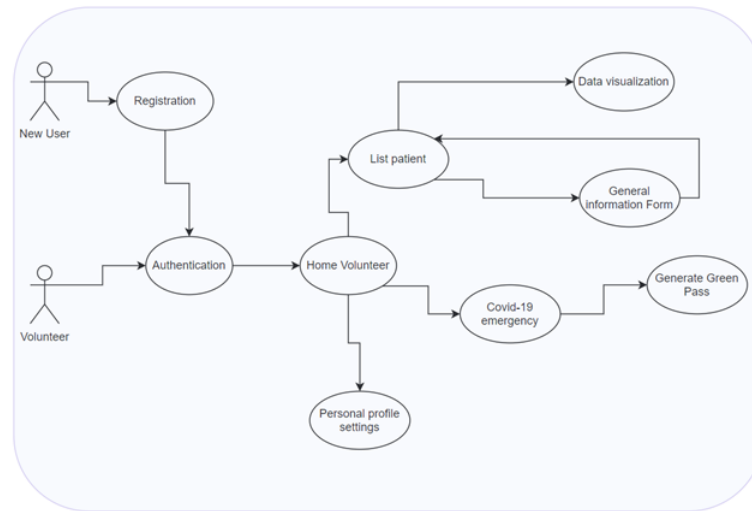


Figure 2.3: Use Cases Volunteer

In this diagram, there is the volunteer flow in the web application. A new user has to register himself by clicking on “volunteer” and then he will have to compile a short form in order to save his general information like phone number and PEC. After the Sign-up and Sign-in process, the volunteer will have his personal homepage, where he can choose among all functionalities. List Patient allows the volunteer to choose the citizen from a list and complete his general information through a form. In Covid-19 emergency he can print the green pass of citizens and see covid reports.

2.3 Sequence Diagram

In this chapter are reported some sequence diagrams which are very powerful items that are used to manage the application dynamics. Here we have some of the fundamental interactions between the user and the web application. In general, there are actors, in this case Doctors and Volunteers, which are the users who interact with the application. Then, in red, there are the views, which are the classes with the goal of data visualization and data render. Each view has a corresponding class that we can call Controller class, which implements the business logic of the application, so it handles algorithmic logics and database calls. Sequence diagrams are very useful because they allow to have a general view on the main actions that the application has to support. It is still a design phase, so the functions that are written in diagrams can be defined as pseudo-code which has the goal of representing

the actions without any precise detail. Sequence diagrams help a lot in the communication with the customer.

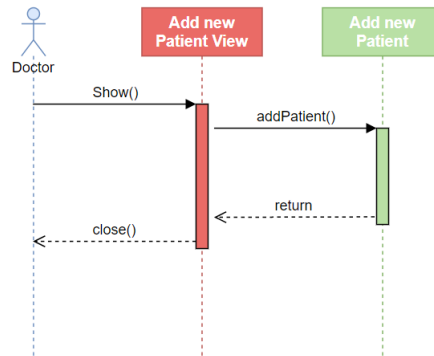


Figure 2.4: add new patient sequence diagram

Here it is reported the most important feature of the web application in a schematic way. It is showed to the doctor the view to add a new patient, in which there will be a form, the more intuitive as possible. Clearly under the view there will be another class which implements the business logic of the application.

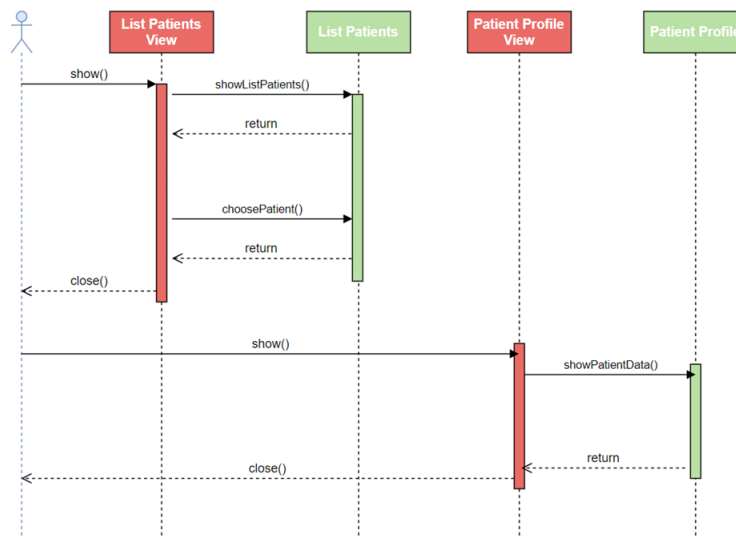


Figure 2.5: Patient Profile visualization sequence diagram

Here it is reported the sequence diagram of the data visualization of a

patient. First of all, the application shows to the user a list of patients relative to the doctor with some basic information. Then the doctor can choose one of the patients and asks for the complete profile. This is handled by *Patient Profile View* and its relative controller *Patient Profile*.

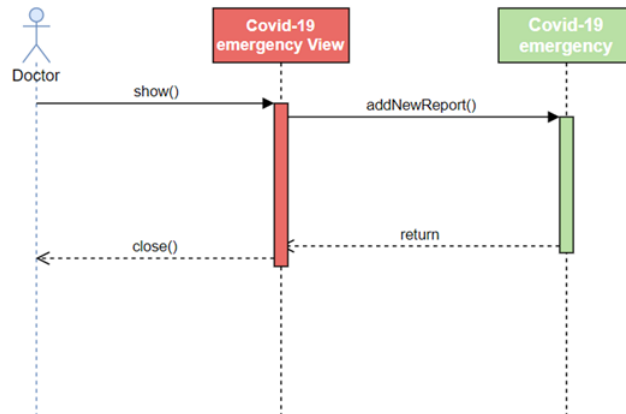


Figure 2.6: Covid-19 emergency sequence diagram

The diagram is showing a simple procedure where the doctor adds a new covid test report. It can be either a covid test or a vaccination and it will be saved in the database. The application will allow also to retrieve a list of tests and vaccinations which are made by a patient. As an example, it is only showed the adding procedure.

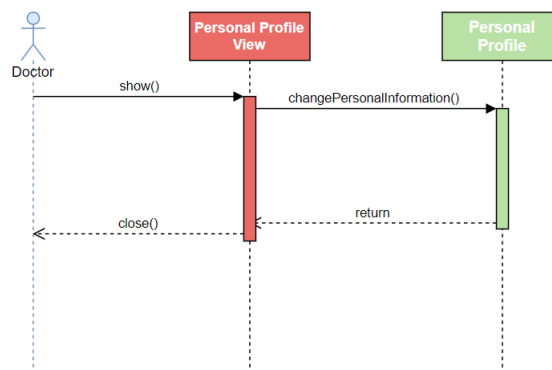


Figure 2.7: Personal profile sequence diagram

In *Figure 2.7* is reported a simple diagram that shows the procedure in order to change personal profile information. The doctor will see his data and he is allowed to change his personal information and register changes.

Moving towards volunteer side, the web application provides functionalities similar to the doctor one. Below are reported some actions that volunteers can make that have significant differences respect to the doctor ones.

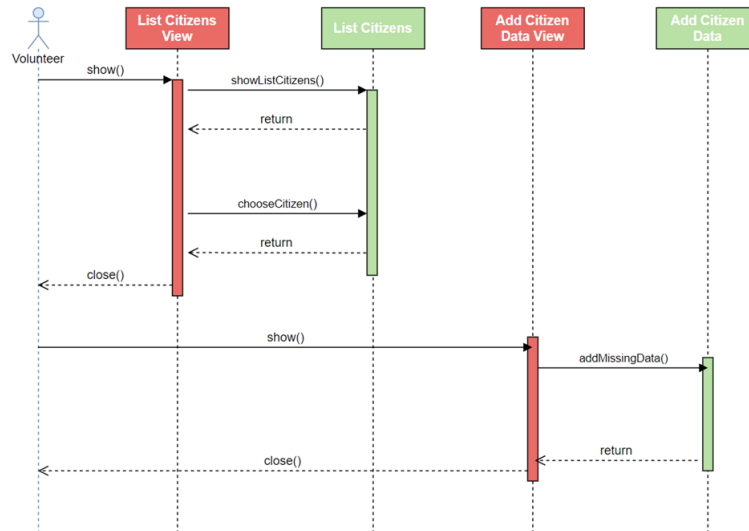


Figure 2.8: Add Citizen sequence diagram

As can be seen in *Figure 2.8*, a volunteer can access the list of citizens registered in the application and choose from the list a citizen. Then, if there are missing data, the web application will show a form to fill in in order to complete the citizen profile with missing information.

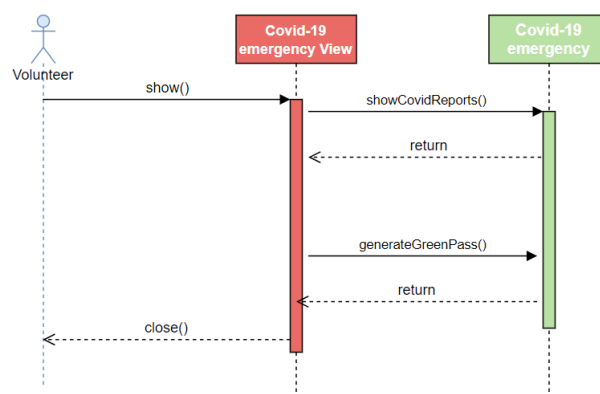


Figure 2.9: Covid-19 emergency volunteer sequence diagram

Here it is shown how volunteer can interact in the Covid-19 emergency

section. He can access to all the covid test and vaccination reports, and also generate the green pass. The Green Pass, according to the rules of the state of Italy, can be generated if the citizen has either a complete vaccination against Covid-19 or a negative test within the last 48 hours. The application should check also these requirements before generating the Green pass.

Chapter 3

Architecture Design

This chapter has the goal to give an overview of the architecture of the application Digital Healthcare. It will cover both the web application made by me and a bit of explanation of the mobile application implemented by my colleague Andrea Calici.

3.1 Architecture Overview

The two applications made by me and my colleague Andrea Calici are completely independent from each other. Both me and Andrea had the possibility to choose the technologies we thought would be better for our own application and none of us was influenced by the choice of the other. The only common point between the two projects was the database. The web application made for family medicine physician and the mobile application made for citizens are completely separated except from the database which is in common. The web app registers data patients and all the information in the database, while the mobile application reads all the data from the database.

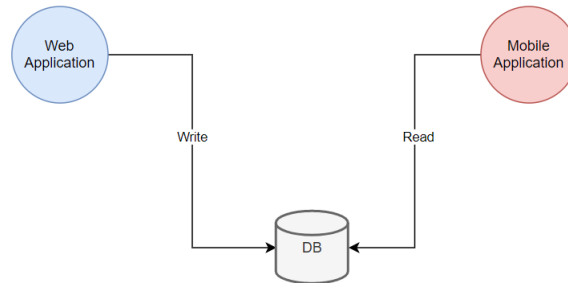


Figure 3.1: General architecture

Talking about the web application, my choice was to use *Angular framework* [6] for the client side of the application, while we agreed that *Google Firebase* [7] was the best database solution for our project. As a matter of fact, before choosing the personal architecture, me and Andrea Calici have discussed about all the database fields that were in common in the two applications. It was very important to agree everything before starting to program because it was the only part that links the web application and the mobile one. The web application is programmed using Angular framework. This framework includes the use of different programming language. The front-end part is mostly written in HTML and CSS with the large use of *typescript* [8] to connect the model part to the view one. As it was said before, the server side is handled by Firebase Database and its features. The authentication is managed by Firebase Authentication [7] which guarantees a safe way to access with email and password to the applications. The images are stored using Firebase Storage [7] and all the data are registered in the database by Firestore Database [7]. The mobile application is written using Flutter SDK [9] which is a relatively new open-source framework made by Google to build user interfaces for mobile and web devices.

3.2 Angular Framework

Angular is a modern development framework which allows to develop across all platforms. It is mainly used for desktop and web development focusing on the front-end part of the application. It uses HTML, CSS and Typescript as programming languages. Angular itself was written in Typescript. Angular benefits are different, the most important are:

- It gives the application a clean and loosely coupled structure.

- It is very easy to test.
- It includes a lot of reusable code.

3.2.1 Typescript

Typescript can be defined as a superset of Javascript, every Javascript code can be converted in Typescript, but Typescript has some additional features. Browsers do not support Typescript, so there is a compiler that each time the application is compiled, it translates Typescript code in Javascript one. In Typescript there are object-oriented features which are not present in Javascript. There are concepts of classes, fields, properties, private and public fields, interfaces which are typical of an object-oriented programming language [10]. In addition, we can catch errors at compile time and correct them before executing the program in the browser. In Typescript there is the concept of Modules. Modules are classes and cohesive block of code which can be exported and imported in other part of the Angular application. Another essential entity is the Component. Each angular app has at least one component that is the root component. Components define views that are the parts that the user sees and which communicate with program logic and data, and they use services which are injected in the components as dependencies making the code more modular. Examples of services are database classes or functions that are used more than one time. Each component encapsulates:

- *Logic*: the methods and the operations used inside the class.
- *View*: the part that is showed to the user.
- *Data*: all the data and information used in the application.

Angular uses Decorators to markup the classes with some metadata that allow the compiler to know which type of class (view, service, . . .) it is dealing with. For example, a component is marked with the decorator `@Component` while a service is marked with the `@Injectable` token. The power of Angular framework is the use of Dependency Injection to inject in the classes some services or pieces of code written in other part of the application. This allows to reuse code and make the application lighter. Another important element is the Template which is a form of HTML that tells Angular how to render the component. The goal is to render in a view the data and the logic of a component. This is done using HTML language and using data binding provided by Angular. Moreover, Angular framework organizes its classes in

a very smart and modular way. When a component is automatically generated by the programmer using angular CLI, that is the command prompt of angular framework, Angular generates a directory with the name of the component in which there are prepared four files:

- *CSS file*: it is used to handle the format of the template using CSS language.
- *HTML file*: it is used to render the view of the component. Using Ng directives, the view part can be bind to the Typescript file in which is encapsulated the logic and the data of the component.
- *Typescript file*: it is the main class of the component in which is contained all the logic of the component including methods and functions. Moreover, it contains the data of the program.
- *Test file*: it is used for test purpose. It allows to test the single component in order to guarantee modularity.

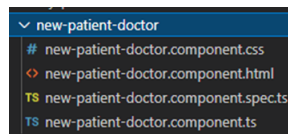


Figure 3.2: New Patient Component

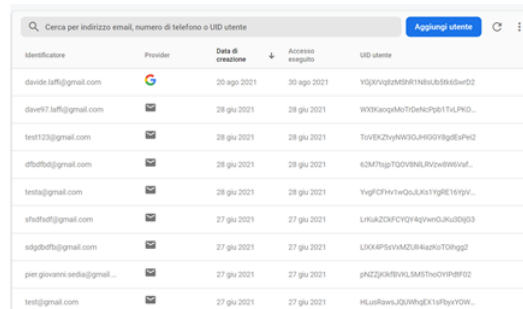
3.3 Server side: Firebase

The server side of the application is entirely hosted by *Google Firebase* [7] and it is divided in a component for the authentication called *Firebase Authentication*, another component called *Firebase Storage* with the goal of storing images and the last one, that is the one which has the function of a Database, called *Firestore Database*, which organizes everything stored in the database in collections and documents. *Firebase* is a platform oriented to the creation of web and mobile applications. It is developed by Google and it offers different services which can help developers and company to create their application and boost them safely. *Firebase* gives the possibility to access to backend services which have the goal to store data inside the Database. *Authentication*, *Storage* and *Firestore Database* are all belonging to this category and, in addition to them, *Firebase* offers other backend services like *Machine Learning* and *Cloud Functions*, which are server side

functionalities which can be very useful for applications. On the other hand, there are monitoring and analytics features that a user can take advantage of like *Crashlytics* and *Performance*.

3.3.1 Firebase Authentication

In applications the authentication phase is crucial. First of all for security reason, but also for personalization. As a matter of fact, is very important to control the access to an application in order not to allow someone not registered to enter the system. Secondly, knowing the identity of a user is essential for Digital healthcare, because each user has a personalized page when he logs in. When a doctor sign in, he must have the doctor's homepage and not a municipal volunteer one and vice versa. Moreover, each doctor and each volunteer needs to visualize his personal list of either patients or citizens and cannot access to other lists different then his own one. Firebase Authentication [7] propose a variety of methods for authentication. It allows the programmer of the application to implement in a very simple way most of the common authentication possibilities. Starting from the classical ones like *Username and password*, *Google and Facebook*, to more particular ones like *Github* and *Yahoo*. The integration of the authentication is very easy and it is completely guided by the website of Firebase. Focusing on Digital Healthcare, there are two authentication methods provided by the web application: email and password, and Google sign In. If a user tries to open the web application while he is not logged in, he will be moved to the login page. If he is not registered, the user can choose to sing up using either email and password by pressing "Registrati" button, or Google authentication by clicking "Login on account google". The Google method is very quick, and the user is asked only to login to his Google account and agree to provide some information. While choosing the email and password authentication, the user has to specify some personal information like name, surname, phone number and some others, and then provide an email and a password. When he finishes to fill in the form, it will be sent a confirmation email to verify the identity of the user. Each user registered in the application has a UID assigned. It is a unique alphanumeric code that identifies each user in the database. In this way, during the programming phase each page can be personalized depending on the user who is logged in. Google authentication was inserted in Digital Healthcare to speed up the authentication procedure, because some data like profile image, phone number and email are automatically filled in because incapsulated in Google's account.



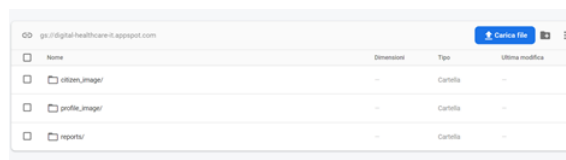
Identificatore	Provider	Data di creazione	Accesso eseguito	UID utente
davide.laff@gmail.com	Google	20 ago 2021	30 ago 2021	YQ3Vvq3M9R1N8uB5Sd8w02
dave97.laff@gmail.com	Email	28 giu 2021	28 giu 2021	W9KaopMo7D6hcPj81T4PK0...
test123@gmail.com	Email	28 giu 2021	28 giu 2021	ToV6KZhyNW3DjH00YtjgSuPh02
d1bdfb@gmail.com	Email	28 giu 2021	28 giu 2021	eDM7tjgTQ0V8N4Rzwb898af...
testa@gmail.com	Email	28 giu 2021	28 giu 2021	YqgCFHvHwD6uJkx1Y9RE18Yjv...
s1adfa@gmail.com	Email	27 giu 2021	27 giu 2021	Lm6KZ3FCYQ4qPw0Jku0Dj63
s1q2bdf@gmail.com	Email	27 giu 2021	27 giu 2021	L00k4P5vYAGUR4uz0T0hgg2
pier.governi.seda@gmail.com	Email	27 giu 2021	27 giu 2021	pNZZq3R6V6L5M5T0v0V98F02
test@gmail.com	Email	27 giu 2021	27 giu 2021	Hu5R6w5JQWwqX1a1byY0V0W...

Figure 3.3: Authentication

In *Figure 3.3* it can be seen a shortcut of the Authentication's part of Digital Healthcare in Firebase. The icon with the small envelope represents users registered with email and password, while the google icon indicates people logged in using Google Authentication.

3.3.2 Firebase Storage

Firebase Storage is a service offered by Firebase used to store images and videos on the cloud. Firebase Storage stores files in a Google cloud storage bucket, making them accessible through both Firebase and Google Cloud.



Nome	Dimensione	Tipi	Ultima modifica
citizen_image/	-	Cartella	-
profile_image/	-	Cartella	-
reports/	-	Cartella	-

Figure 3.4: Storage

In *Figure 3.4* are reported the folders that are present in the storage part of the web application. Firebase storage [7] is used to store files and it has the possibility to create folders to better organize users' images and files. First of all, Firebase needs to be integrated in the application. Then, you need to reference the path of the file you want to upload or download depending on either where you want to save it in the cloud or from where you want to download it. For example, indicating a path to upload a file as "citizen_image/xxx.png", it will be uploaded the image "xxx.png" in the folder "citizen image". Talking about Digital Healthcare there are 3 folders. Citizen_image contains the pictures of all the citizens who have changed their profile pictures. When a municipal volunteer, in the changing data section, decides to upload a new profile picture for a citizen, it is directly uploaded

in that folder in the cloud. The name of the picture will be the fiscal code of the citizen followed by the extension of the image. If the user has already a profile picture and the volunteer uploads a new one, the newer image will overwrite the older one. Then there is the Profile_image folder which collects all the profile images of doctors and municipal volunteers registered to the web application. As it is done for the citizen images, each picture uploaded to this folder is saved with the fiscal code of the user followed by profile and the extension of the image. The last folder is used to store reports, which are PDF files that contain COVID-19 reports like tests and vaccination. These files can be uploaded by doctors and are different for all the patients. Then these reports are downloaded by municipal volunteers when they need to visualize documents related to COVID-19. Each PDF file is saved with the fiscal code of the citizen followed by the word “report” and the timestamp in which the report was uploaded to keep it unique.

3.3.3 Firestore Database

Firestore Database [7] is a NoSQL database hosted on the cloud. It introduces an important change in database structures, organizing data in collections and documents. An admin can create some *collections* and each one can be populated of independent *documents* which have some attributes called *fields*. Documents can support different data types: number, strings, dates, and others. Moreover, a document can contain another collection creating nested objects and structures very useful.

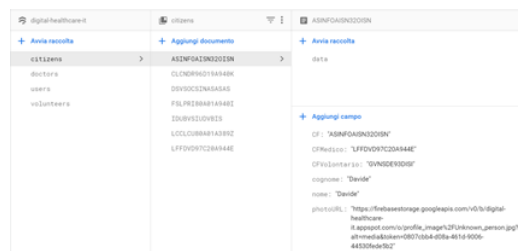


Figure 3.5: General architecture

In *Figure 3.5* it is showed the general architecture of Digital Healthcare. The database is in common with the mobile application made by Andrea Calici. There are 4 different collections which encapsulates all the information of our application:

- **Citizens:** it contains all the information about patients and citizens added by doctors and volunteers inside the database, both Covid-19 data and general patient data.

- **Doctors:** it groups together all the doctors registered to the application.
- **Users:** it is an “helping” collection that simplify handling users registered to the application, both doctors and volunteers, holding general and most used fields of users.
- **Volunteers:** similar to the collection Doctors, it groups together all the volunteers registered to the application.

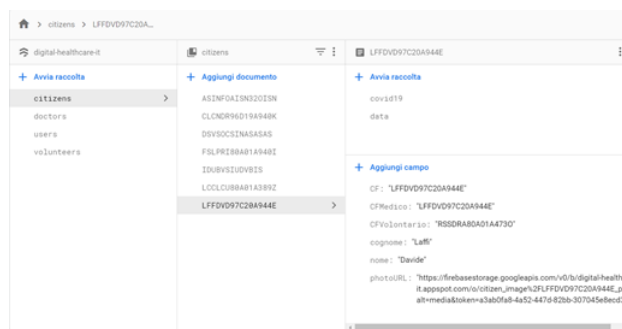


Figure 3.6: Citizens collection

Citizens’ collection is the most important in the database because it contains all the information of a citizen. Each document is identified by the fiscal code of the citizen. As it can be seen in the right side of *Figure 3.6*, each citizen has some general attributes like name, surname and the fiscal code of the referenced doctor and volunteer which have inserted the data and two collections called Covid19 and data.

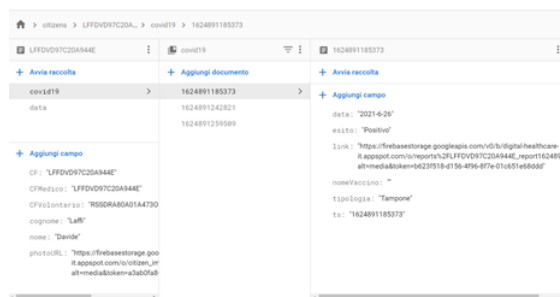


Figure 3.7: covid19 collection

Covid19 collection has a document for each report uploaded by the referring doctor identified by the timestamp at which it was uploaded. Then

each document has some fields with the information related by the report. In *Figure 3.7* it is shown the example of a Covid test, so there is the date on which the test was performed, the outcome of the test, the link to the report and the type of the test.

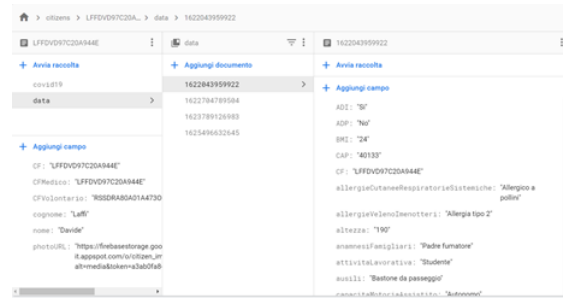


Figure 3.8: data collection

Data collection has a document for each time a new group of data is added by a doctor. As a matter of fact, each document is identified by the timestamp representing the instant on which the group of data was added. For each timestamp there is a bunch of fields which contains all the information of a citizen added both by the relative doctor and the relative volunteer. All the data are registered as strings and are casted in the real type of data when they are either wrote or read by the database.

Chapter 4

Implementation

This chapter has the goal of illustrating the implementation of the web application, starting from the design part up to the coding phase. It will be analysed the designing of the packages and the classes that are needed to implement the whole system, but also some interesting coding part which represents key points of the business logic of the application.

4.1 Packages diagram

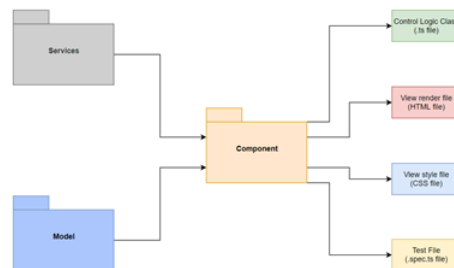


Figure 4.1: Packages diagram

The macro categories in which the project can be divided are:

- Model
- Services
- Component

In general, the system is composed by a variety of components which implement all the functionalities of the application, they render all the views with whom the user interacts with, and they integrate all the external services that the application needs to work. The cardinal package is the component which together with the others implements all the logic of the application. Each component needs to exchange information with the Model which is a package that contains all the plain objects that are used inside the application. The model class is called to create an object already defined in the past, to facilitate the modularity and reusability of the application, the object is defined in the model together with its fields. Another package very important is the Services package. A service is a narrow class which should do something specific and should not have more than a purpose [6]. The power of services is that components can fully concentrate on the user experience, connecting the view part and the data logic of the application, and they can delegate to services some tasks. To enable the communication between components and services, Angular uses the dependency injection. In this way, when a component needs a specific service, it is injected in the constructor of the class and it is used by the component. There are different types of services, starting from database methods, but also authentication' services and routing protection' services.

4.2 Model

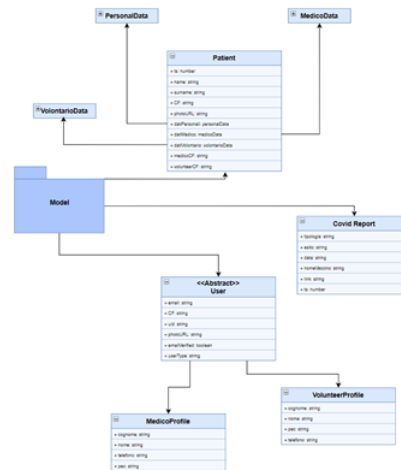


Figure 4.2: Model diagram

In *Figure 4.2* is reported the diagram of the model. The package contains all the classes belonging to the model part of the application. The classes represent plain object which have only the definition of the fields and the constructor.

```
export abstract class User {  
  
    public email: string;  
    public CF: string;  
    public uid: string;  
    public photoURL: string;  
    public emailVerified: boolean;  
    public userType : string;  
  
    constructor () {  
        this.email = "";  
        this.CF = "";  
        this.uid = "";  
        this.photoURL = "";  
        this.emailVerified = false;  
        this.userType = "";  
    }  
}
```

User is an abstract class. There are six public fields with the type of object defined. Then there is the constructor which is used to initialize the object. In the model of the application there are three main objects which are *Patient*, *User* and *Covid Report*. Starting from user, it is an abstract class because, as it was said before, there are two types of users which can register to the application: doctors and volunteers. So, the decision was to create an abstract class for the entity user, and then two classes that are *MedicoProfile* and *VolunteerProfile* which extend the superclass. This choice guarantees to avoid duplication of code, the alternative was to create two classes which do not extend any class, and so to duplicate the common part of the two classes which is the one in class user. Moreover, the creation of the abstract class allows to add new types of users if future improvements require it. Because every user in the application must have the fields of the class user declared, then it can modify the other fields depending on the type of user it is. Patient is an object which encapsulates all the fields of a patient that is registered to the system. The name Patient should be citizen, because it includes both the medical attributes of a patient inserted by the

doctor, and the more general data inserted by the volunteer. Because of the multiplicity of data, it was chosen to create other three “helping” objects to facilitate the application modularity from a coding point of view, but also to organise better the division of fields. The three objects are:

- **PersonalData:** it includes all the attributes defined as general data of the citizen. For example, gender, date and place of birth, phone number and email.
- **VolunteerData:** it covers all the information inserted by the municipal volunteer in the web application. For example, ID card, ICE numbers and other codes.
- **MedicalData:** it encapsulates the data registered by the family medicine physician. For example, relevant diseases, allergies and other.

In addition to these three objects, the class patient has some general attributes which are required to define it. The most relevant are name, surname, but also *medicoCF* and *volunteerCF* which indicates the fiscal code respectively of the doctor and of the volunteer which have inserted the data of the patient. This is done to track all the citizen and to show to doctors and volunteers only their patients and citizens based on the fiscal code. The last object that is present in the application’s model is *Covid Report*. It is an object relative to covid19 emergency part of the system, and it is used when a user either registers or wants to read a Covid report, like tests or vaccinations. The fields of the object are:

- **Tipologia:** which indicates the type of Covid report it was uploaded. The value it can assume are “Tampone”, “Vaccinazione”, “Sierologico”.
- **Esito:** in case of a Covid test, like “Tampone” or “Sierologico”, it will have the result of the test as a string. Otherwise, in case of vaccination, it will have the step of vaccination, like “Prima dose”.
- **Data:** it indicates the date on which the report was realized.
- **nomeVaccino:** it is a blank field in case of covid test, while it contains the name of the vaccination, in case that the report is a vaccination one.
- **Link:** it contains the link to the report saved on the Storage of the database.
- **Ts:** it indicates the timestamp on which the Covid report was uploaded to the database.

4.3 Main classes

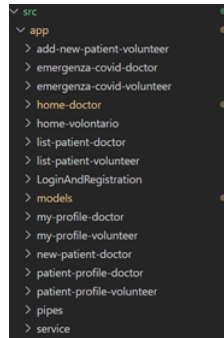


Figure 4.3: Packages list

In *Figure 4.3* there is a list of all the packages and component which compose the application. Model is the package that contains the classes illustrated in chapter 4.2. Then there are pipes and service which are packages that will be discussed in the next paragraphs. The others are component. Each component has a Typescript file, a HTML file, a CSS file, and a test file. It will follow a list of the most important components of the application with a description of the features they introduce and some screen of the most relevant part of code. For the majority of components, they are duplicated because the system implements some similar features both for doctors and for volunteer in a decoupled way. For example, both doctors and volunteers need to add citizen data, but the two components cannot be joined together.

4.3.1 Login and registration

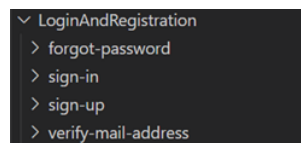


Figure 4.4: Login and registration package

The *login and registration* directory contains four components. The registration process is implemented by the component *sign-up*. The aim of the class is to realize all the user's registration phase. There is a form which asks the user to fill in his personal data, starting from if he is a doctor or a volunteer. Then data will be registered in the database, and it will be sent a verification mail to the one inserted in the form.

```

async onSignupMedico(){
    this.medicoCreated = new MedicoProfile();
    this.medicoCreated.userType = "medico";
    this.medicoCreated.CF = this.CF;
    this.medicoCreated.cognome = this.cognome;
    this.medicoCreated.email = this.email;
    this.medicoCreated.nome = this.nome;
    this.medicoCreated.pec = this.pec;
    this.medicoCreated.telefono = this.telefono;

    try {
        await this.firebaseService
            .signup(this.medicoCreated, this.password);

        if(this.firebaseService.isLoggedIn())
            this.isSignedIn = true;
    } catch (error) {
        window.alert(error);
    }
}

```

This piece of code is extracted from the signup part. The method `ngOnInit` is an angular method that is called at the beginning of the process and the class has to implement the interface `OnInit` to use it. At the beginning the program controls if the user is already signed in and it sets a Boolean value used to check it. The main method is `onSignup` which divides doctor's requests from volunteer's one calling the specific method. By checking the string variable `userType`, the method calls either `onSignupMedico` or `onSignupVolunteer`, respectively if the user specified either doctor or volunteer. The method simply creates and instantiates an object that can be `MedicoProfile` or `VolunteerProfile` and, through angular data binding, it collects data from the form implemented in the HTML page. Then, with a try catch statements, it calls a service that was previously injected in the constructor of the class called `AuthService`, which has some useful tools to handle the authentication of users. One of the service's method is `signup`, that asks as parameters either the doctor or the volunteer profile object and the password inserted in the form and connecting itself to firebase authentication service, it register the user to the database.

```

<div class="signup-form">
  <form (submit)="onSignup()" ngNativeValidate
  #formSignUp>
    <div id="zeroSection" [ngClass]="
    {'non-display-section':controllerSection!=1}">
      <div class="form-check vertical-padding10">
        <input class="form-check-input" type="radio"
        name="userType" id="medico" value="medico"
        [(ngModel)]="userType">
        <label class="form-check-label"
        for="flexRadioDefault1">
          Medico
        </label>
      </div>
      <div class="form-check vertical-padding10">
        <input class="form-check-input"
        type="radio"
        name="userType" id="volontario"
        value="volontario"
        [(ngModel)]="userType">
        <label class="form-check-label"
        for="flexRadioDefault2"> Volontario
        </label>
      </div>
      <button type="button"
      class="btn btn-lg btn-primary"
      (click) = "changeSectionGo()">
      Inizia Registrazione </button>
    </div>

```

Some relevant keywords are *ngClass* which is an Angular command that is used to hide or use a class depending on a true or false condition. In the web application, this feature is often used to display some part of the page depending on some condition. For example, if a form is composed of more sections and it's needed to display only one of them at a time, with *ngClass* and a counter the other sections different from the counter can be hidid. Another important angular keyword is *ngModel* which is used for data binding. It is very powerful because it can link together part of the HTML variables and form parts to the Typescript variables value. In this way the Typescript class and the HTML one can communicate in a

bidirectional way. As it can be seen in the snap of code above, each input tag is bind together with a variable of the typescript class. So, when a user fill in the form, all the values he is inserting are immediately transmitted to the logic part of the application handled by typescript. The last command is *ngNativeValidate* which is used to check the integrity of the form. This instruction automatically activates the check of all the fields with a required keyword in. Talking about the sign in part, the code is smoother. A form implemented in the HTML page is displayed to the user and by fill in it and pressing a button, the typescript class makes use of the authentication service injected in the constructor and asks Firebase to log in the user with email and password. If the user is already registered, the server gives a positive response to the client and set a Boolean variable to true, allowing the user to enter the application. Otherwise, an error message is displayed to the user. There is another method in the sign in class which is the google authentication one. Also for this one, the class uses the authentication service calling the relative function of firebase.

4.3.2 Home

There are two different components respectively one for doctors and one for volunteers which implement the view and the logic of the homepage.

```
ngOnInit(): void {
  const userToConvert = localStorage.getItem('user');
  this.currentUserRaw = userToConvert !== null ?
  JSON.parse(userToConvert) : "";

  const documentRef = this.firestore
  .collection('users').doc(this.currentUserRaw['uid']);
  const collectionInstance = documentRef.valueChanges();
  var subs = collectionInstance.subscribe(ss => {
    this.currentUserTemp = this.db
    .createUserMedicoFromSnapshot(ss);

    const documentRefMedico = this.firestore
    .collection('doctors').doc(this.currentUserTemp.CF);
    const collectionInstanceMedico = documentRefMedico
    .valueChanges();
    var subs2 = collectionInstanceMedico
    .subscribe(ss => {
```

```

    this.currentUser = this.db.
    createCompleteMedicoFromSnapshot(ss , this.currentUserTemp);

    localStorage
    .setItem( 'medico' , JSON.stringify(this.currentUser));
    localStorage
    .setItem( 'medicoCF' , JSON.stringify(this.currentUser.CF));
    this.mySubs.push(subs2);
  });
});

this.mySubs.push(subs);
}

```

The snap of code represents the constructor and the initialization method of the homepage class. In the figure there is the doctor homepage, but it is similar also for volunteer one. As it can be seen in the constructor there are some services injected that are used inside the logic of the class:

- **AuthService**: as it was said before, it is used for authentication, in this specific case it is used to allow the logout of the user when a button is pressed.
- **AngularFirestore**: a very important service in the application. It is used to communicate with Firebase database. By invoking the service and accessing to it, there is the possibility to explore and get collections, documents, and snapshot of the database.
- **DatabaseService**: it is an “helping” service where there are some useful transformation methods, in particular when some data are asked to the database and must be converted in typescript objects.
- **NgbModal**: which is used to create modal form.
- **Router**: the routing module is an Angular service used for navigation among the different pages of the application.

Then there is the initialization method. First of all, it uses the local-storage variable that is a default Angular keyword which allows to access to an in-browser storage that has allocated space for each domain. During the login phase, the service stores in the storage an item that can be retrieved with the string user. In this way, even if the screen and classes changed, the

values of the item “user” can be obtained by invoking `localStorage` and using the method `getItem`. Data in `localStorage` are handled like a JSON dictionary. Then the class has to create the `MedicoProfile` object. To do that, the `AngularFirestore` service is used and from the database all the data relative to the user are taken. Then, the method `createCompleteMedicoFromSnapshot` of the class `DatabaseService` converts firebase snapshot into typescript object and it is stored in the `localStorage` in order to simply have it in all the others part of the application until the user connected either presses the logout button or closes the browser. There is another important feature handled by the home component which concerns the functionality which allows to add a new patient to the database. This functionality is dedicated only to doctors, because municipal volunteers do not need to add new citizens, but only to complete the profile of a patient previously inserted. When the family medicine physician wants to add a new patient, it is displayed a modal which asks for the fiscal code of the patient. Here we have different scenarios that can happen:

- **The fiscal code is new to the database:** In this situation, the route navigation brings the user to the Add new Patient section, so that he can proceed to add patient features.
- **The fiscal code is already present in the database:** The home view shows the last date update for the registration of data of this particular patient. The user can choose either to start to add the patient from the last updated data or to add the patient from scratch.

4.3.3 Add new Patient

This component implements the most important feature in the whole system, that allows to start the creation of a new citizen inside the database. The terminology is the same, citizen is used when we are talking about a volunteer who is adding a new person, patient is relative to a doctor adding the person's data.

```
ngOnInit(): void {
  //Take CF from home
  const codiceFiscaleRaw = localStorage.getItem('CF');
  this.CFfromHome = codiceFiscaleRaw !== null ?
  JSON.parse(codiceFiscaleRaw) : "";
```



```

localStorage.removeItem('CF');

//Take if it is the first time you insert the patient
const firstTime = localStorage.getItem('firstTime');
this.isFirstTime = firstTime !== null ?
JSON.parse(firstTime) : "";
localStorage.removeItem('firstTime');

//Take if you are starting from a different ts
const lastUserData = localStorage
.getItem('lastUserData');
localStorage.removeItem('lastUserData');

//Take Medico CF who will add data
const medicoCFraw = localStorage
.getItem('medicoCF');
this.medicoCF = medicoCFraw !== null ?
JSON.parse(medicoCFraw) : "";
localStorage.removeItem('medicoCF');

this.patient.CF = this.CFfromHome;
console.log(this.CFfromHome);

if(!this.isFirstTime){
  const collectionRef = this.firestore
  .collection('citizens').doc(this.CFfromHome);
  const collectionInstance = collectionRef
  .valueChanges();
  var sub = collectionInstance.subscribe(ss => {
    this.patient = this.db
    .createGeneralPatientFromSnapshot(ss);
    this.mySubs.push(sub);
    if(lastUserData !== null){
      this.lastUpdateTS = lastUserData !== null ?
      JSON.parse(lastUserData) : "";
      const collectionRef = this.firestore
      .collection('citizens').doc(this.CFfromHome)
      .collection('data').doc(this.lastUpdateTS+"");
      const collectionInstance = collectionRef
      .valueChanges();
      var sub2 = collectionInstance

```

```

        .subscribe(ss => {
            this.patient = this.db
                .createPatientFromSnapshot(ss, this.patient);
            this.mySubs.push(sub2);
        });
    }
    });
}
}
}

```

The snap of code belongs to *add new patient* component. At the beginning it retrieves from the *localstorage* some variables which are needed to understand in which scenarios are we. The value “firstTime” is a Boolean used to know if, in home component, the user has inserted a new fiscal code which is not already registered in database. In this case the creation of the new patient should be from scratch. Then, if the user has chosen to start adding patient information from the last updated date, the component takes the value of the last date from the localstorage and it makes an access to the database collecting all the data. Then, it transforms snapshots in typescript objects using the DatabaseService, and through data binding, it displays to the user the standard form for a new patient, but with the fields already filled with last data available.

4.3.4 Patient profile

The other two relevant components are the ones related to the patient profile, both from the volunteer point of view and from the doctor one. The goal of the component is to retrieve data from the database of a patient selected from a list and to display the information to the user. The important feature introduced in this component is the possibility to change the date on which where inserted the data. As it was seen in the database explanation part, each patient has a collection called data, and for each collection there are different documents identified by the timestamp on which data where inserted. In this way, for the same patient there are more than one collection of data depending on the date they where added. Therefore, the system allows to have some statistics of the patient, because, in the database it is registered all the timeline of him.

```

<div class="dropdown-menu"
  aria-labelledby="dropdownMenuButton">
  <div *ngFor='let ts of arrayDatePatient'>

```

```

        <a [routerLink]=""" fragment=""
        (click)="changeDatePatient(ts)"
        class="dropdown-item">{{ts |
        date:'dd/MM/yyyy'}}</a>
    </div>
</div>
</div>
</div>
</div>
<div class="col-md-5 text-center">
    <h3>Data del: {{lastChosenTS |
    date:'dd/MM/yyyy'}}</h3>
</div>
</div>
</nav>
</div>

```

This is a snap of code of the HTML part which allows to change date of patient's data. There is a dropdown menu which shows all the dates on which data for a specific patient were added. This menu is implemented by the keyword *ngFor* which is an Angular command that allows to replicate the code for all the elements of an array. By choosing one element of the *arrayDatePatient*, the program calls the method *changeDatePatient* and asks to the database all the data related to the specific timestamp. In this way the object changes his fields with the updated data and through data binding, the new information are showed to the user.

4.4 Services

Talking about the services that are used in the application, it was already said what a service is. These typescript classes are marked as injectable because they can be used by other classes through dependency injection. Services need to implement specific features which the main classes delegate to them. In the case of Digital Healthcare, there five relevant services which are used.

4.4.1 Database

The first service is a helping service that is used to simplify the management of the database. The main features are allowing a clean communication between typescript objects and database's snapshot data. When some data

are retrieved from the database, they usually are in a raw form, and it is very risky to use them without modifications.

```

createPatientFromSnapshot(patientFromDB : any,
resultPatient : Patient){

    //Dati personali
    resultPatient.CF =
    patientFromDB.CF
    resultPatient.nome =
    patientFromDB.nome;
    .....

    //Dati Medico
    resultPatient.datiMedico.codiceEsenzione =
    patientFromDB.codiceEsenzione;
    resultPatient.datiMedico.retiPatologieAssistito =
    patientFromDB.retiPatologieAssistito
    .....

    //Dati Volontario
    resultPatient.datiVolontario.numeroCartaIdentita =
    patientFromDB.numeroCartaIdentit ;
    resultPatient.datiVolontario.comuneRilascio =
    patientFromDB.comuneRilascio;
    resultPatient.datiVolontario.dataScadenza =
    patientFromDB.dataScadenza;
    .....
}

```

An example is the piece of code above which shows the main method of the class. *createPatientFromSnapshot* takes as parameter the snapshot just taken from the database, and it converts it to the typescript object of patient. The method is really simple, for each field of the object, it accesses to the snapshot and takes the relative attribute.

```

createMedicoFromStorage(medicoRaw : any){
    var medico = new MedicoProfile();

    medico.CF = medicoRaw['CF'];
    medico.cognome = medicoRaw['cognome'];
}

```

```

    medico.email = medicoRaw[ 'email ' ];
    medico.uid = medicoRaw[ 'uid ' ];
    .....

    return medico;
}

```

This is another method which has a very similar goal to the previous one. The method *createMedicoFromStorage* is used to convert the doctor data saved in the localstorage of the browser to an object.

4.4.2 Storage

This service groups together all the services related to the storage upload part. In the system, when it is needed to upload to Firebase storage, must be called a service which handles the process. There are three possible type of uploads that can be performed inside Digital Healthcare. The first one is for user's profile picture, so when doctors and municipal volunteers want to change their personal profile, another one is for citizen's profile picture, so when a volunteer changes the profile picture of a citizen, while the last one is for storing covid19 report, so PDF files.

```

async uploadFile( nomeFile: string ,
CF: string , dateTS: string , report : CovidReport ) {
  if ( this . file ) {
    console . log ( this . file );
    const filePath = `${ this . basePath } / ` + nomeFile ;
    const snap = await this . storage
      . upload ( filePath , this . file );
    this . getUrl ( snap , CF , dateTS , report );
  } else {
    alert ( ' Please select an image ' );
  }
}

```

This is an example of how the storage upload works [13]. Here it is illustrated the coding part which allows to store in the firebase cloud the file of a *covid19 report*, but also the other upload processes work in a similar way. The first two methods have a preparation role saving the files in the typescript object and setting the path of the storage in which the file is going to be saved. Then there are two async functions which carry out the upload process. The method *uploadFile* is the one that actually performs the upload with the

method `upload` called on the Firebase storage service. While the method `getUrl` is used to save information in the Firestore Database about the report the user is saving. So, while the file is being saved, the method communicates to the database to create a document on the collection “`covid19`” with the timestamp indicating the time on which the report is added and some general information of the report as attributes.

4.4.3 Guard

A quick regression should be done talking about guard [14]. It is a powerful service provided by Angular because it allows to protect the routing system of the application. To implement guard system there should be a service class which establishes the rules to follow and then an annotation on the app routing module next to the routes which should be protected by the guard. In the web application there are not users which have higher privileges respect to others, so the rules implemented are:

- To protect not logged users from entering the application, so that a user that is not logged in cannot reach part of the application different then the login and registration pages.: In this situation, the route navigation brings the user to the Add new Patient section, so that he can proceed to add patient features.
- To differentiate municipal volunteers’ part from doctors’ part, so that a doctor cannot access to volunteer pages and vice versa.

4.4.4 Pipes

The last service that will be mentioned is Angular Pipe service. Pipes are simple functions which are used in template expression, so in the HTML file of the component. They take as an input values and they give as an output transformed values. Pipes are very powerful because they can be declared only once in the program, but they can be called different times. There are some pre-crafted pipes that angular provides, for example *DatePipe* which formats a date value, *UpperCasePipe* which puts every character as input to its uppercase, and others. The developer can create his own pipe and use it as in this system.

```
<div class="col-md-8" #tampone>
  <div class="card vertical-padding10"
    *ngFor='let rep of report |
    tipologiaFilter:tipologia'>
```

```

<div class="card-header">
  <div class="row">
    <div class="col">
      {{rep.data}}
    </div>
    <div class="col text-end">
      
    </div>
  </div>
</div>

```

This is an example of the use of the *tipologiaFilter* pipe taken by the HTML class *emergenzaCovidDoctor*. It is a service that takes as input an array of reports, which is an object that represents Covid19 report which can be either a Covid test or a vaccination result, and a string called “*tipologia*” which can be one of the three covid report typology. The pipe gives as output all the items of the array which match the typology searched by the user. So, if the user chooses vaccination as typology, the array is filtered and all the items that have not “vaccination” as attribute are not shown.

```

transform(items: any[], searchText: string,
showAll: boolean, currentUser: MedicoProfile
| VolunteerProfile): any[] {

  if (!items) {return [];}

  if (!showAll) {
    if (currentUser.userType == "medico") {
      items = items.filter(it => {
        return it.CFMedico == currentUser.CF;
      })
    }
    else if (currentUser.userType == "volontario") {
      items = items.filter(it => {
        return it.CFVolontario == currentUser.CF;})
    }
    if (!searchText) {return items;}
  }

  searchText = searchText.toLocaleLowerCase();

```

```
return items.filter(it => {
    return (it.nome + " " + it.cognome + " " +
        it.CF).toLocaleLowerCase()
        .includes(searchText);});
}
```

This is the pipe *tipologiaFilter* we were talking about. As it can be seen, pipes are marked with the decorator *@Pipe* and the name of the pipe. Then, with the marker *@param*, the parameters are defined. In the method *transform* there is the real filtering process with the method *filter* called on the input array. The other pipe it is implemented in the application is another filtering pipe which takes an array of patients as input. The goal of the pipe is keeping only the elements of the array which were inserted by the user, so that if the user is a doctor, the item should have the doctor's fiscal code as *CFmedico*, while if the user is a volunteer, the item should have the volunteer's fiscal code as *CFvolunteer*. The last pipe used in the system is the *DatePipe* which is able to transform a timestamp to another date format.

Chapter 5

Functionalities and demonstration

In this chapter, it will be shown a demonstration of the key points of the application working. The system is complete, and all the features are fully implemented.

5.1 Login and registration

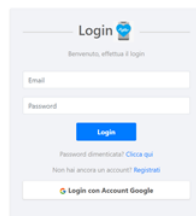


Figure 5.1: Login page

When the user searches on the web the web application Digital Healthcare, the first page that is shown from the application is the Login page. This is because the system does not allow to user not logged in to access to application's functionalities. If the user is already registered to the database, he can now choose to sign in by using email and password, otherwise he can choose google sign in method. If this is the first time that the user faces the login screen, he can click on register and move to the registration phase. The last option is reserved to users who either lost or forgot their password. By clicking on "password dimenticata" they will be able to recover the password by inserting their registration mail.

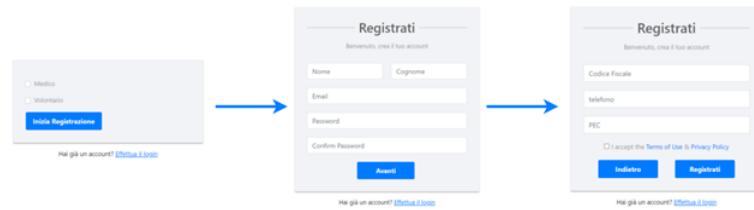


Figure 5.2: Registration phase

In *Figure 5.2*, it is depicted the flow of a user, who wants to sign up to the web application for the first time. The first step is choosing the type of user he will be by selecting one out of the two radio boxes. After identifying if the user is either a doctor or a municipal volunteer, the system shows the first part of the registration form, where the user is asked to fill in general data like name, surname, email, and password. The password must be confirmed and the system will allow to press the button to proceed only if the two strings are equals. The second part of the form asks to the user other data and then he can complete the registration process. After that a confirmation mail will be sent to the user mailbox and he will be able to sign in with that email.

5.2 Home

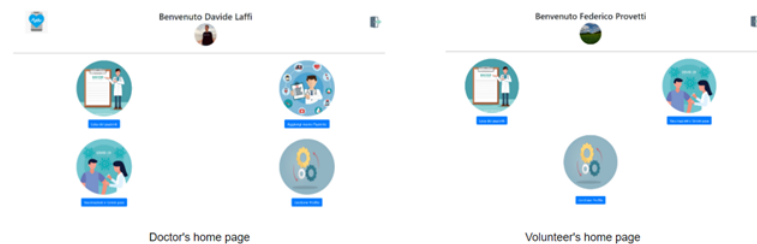


Figure 5.3: Home page

The main page of Digital Healthcare is the home page. It is the first page that is shown to the user after the login, and it contains all the functionalities of the application. In *Figure 5.3*, there are both the doctor's home page and the volunteer's home page, respectively on the left and on the right side. The functionalities are very similar for both the users, but, as it can be seen, the volunteer cannot add a new citizen from scratch from the home page. The functionalities are:

- **List patient:** to have access to the list of patient relative to a certain doctor or volunteer and to visualize citizen's data.
- **Covid19 emergency:** to access covid19 to check all the documents relative to covid tests and vaccinations
- **Profile settings:** to access the user personal page and change user's information
- **Add new Patient (only for doctors):** to add a new citizen inside the database with all the relative data.

5.3 Add new citizen

The process of adding a new citizen into Digital Healthcare is divided in two parts. The first one is performed by the doctor, which can create a new citizen profile into the database and he can insert all the medical data. Then, there is a second section which is volunteer's expertise that allows to complete the citizen profile. The doctor has a dedicated section in the home page called "aggiungi nuovo paziente" which allows to access directly to the form. Instead, the volunteer has to click on citizen list and, by clicking on a person, he will access the form to complete data.

5.3.1 Add new patient by doctor



Figure 5.4: Add new patient

The doctor has the goal of creating the patient's profile for the first time. The first step to add a new patient is in the home page, where the doctor, by clicking on "aggiungi nuovo paziente", will face a small modal form which asks for a fiscal code. Here there are two possible scenarios; if the doctor fills in the form with a fiscal code which is not already saved in the database, the application will allow the user to navigate to the real add new patient page. While if the fiscal code is already present in the database as in Figure

5.4, the application will collect from the system the last date on which data where modified and will show it to the user. Then, the doctor can either click on the date or click on the button “Aggiungi da zero”. The first option will not create another patient inside the database, but it will bring the user to the add new patient page with the standard form already precompiled with the data of that date. While the second option will create, for the same patient, a new document from scratch without data saved under that specific timestamp.

Figure 5.5: New patient form

In *Figure 5.5*, it is provided an example of the form to add a new patient to the database. It has a very simple structure, and it is divided in categories of information. There are six classes of attributes which include some general data of the person and the more medical aspects. The user has to complete all the required fields of the form, otherwise the program blocks the proceed button and it does not allow the user to continue. When the registration button is pressed, the system notifies to the doctor the result of the procedure, and if everything was fine, it moves the user to the home page of the application. Now, in the database there is a new citizen created which is assigned to the doctor by fiscal code. All the fields are created but only those inserted by the doctor are not empty. Then there is a second phase executed by the municipal volunteer.

5.3.2 Add new citizen by volunteer

Talking about the volunteer point of view, the procedure to complete citizen’s data is a bit different. By clicking on citizens’ list the user will see the list of all the citizens which have been added by him with a green tick next to their name to highlight that data are complete.



Figure 5.6: Citizens' list

By tapping on the checkbox “mostra tutti i cittadini”, the web application will show to the user all the citizen registered to the database, adding to the previous list both the citizens assigned to another volunteer, but also those who do not have a referring volunteer at all. The citizens who do not have a volunteer assigned have an incomplete profile, because no one has inserted the missing general data. So, they are put to the top of the list and marked with a yellow exclamation mark. Now the user can click on one of the citizen which are divided in three types:

- **Citizen without volunteer:** the user is moved to the add new citizen page with a form to fill in with all the data relative to volunteer’s part. By clicking on the image on the top of the page, he can also change the profile picture of the citizen. When the process is finished, by tapping on the button at the bottom of the page, the volunteer will register data to the database and complete the profile. See *Figure 5.7*.
- **Citizen with volunteer different from user:** the system shows to the user the citizen’s profile page with all the information.
- **Citizen with the user as volunteer:** the user is able to see the whole citizen’s profile page and to modify data.

Figure 5.7: Citizen’s profile

5.4 Citizen Profile

The citizen profile section is another important part of Digital Healthcare. It can be accessed through the list of citizens from the homepage both from doctors and from volunteers and it is dedicated to the visualization of data of citizens. As in the previous chapters, it will be analysed first of all the doctor side, and then the volunteer part.

5.4.1 Patient's profile



Figure 5.8: Patients' list

In order to visualize patients' information, the doctor should pass through the patients' list that is shown in *Figure 5.8*. The standard list includes only the patients related to the current user, so that each doctor can see only his personal patients. By clicking on the checkbox above, the user can visualize all the patients that are saved in the database of the application, and those who are not related with him will be highlighted in orange. Now the user can choose to click on one of his patient or on another one. In the first case, he is moved to the patient's profile section with the possibility to modify patient's data, while in the second case, he will not be able to change patient's information, but he can visualize data anyway.

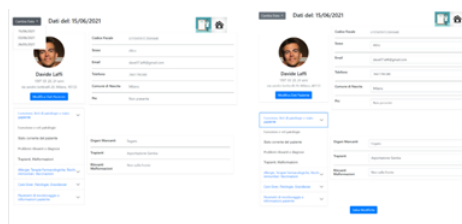


Figure 5.9: Patient's profile

In *Figure 5.9* is shown how patient's profile section looks like. In the left side of the image there is the first visualization of data. In the left part there

is the patient profile with the picture, age and where he lives. Then in the centre-right of the screen there are the more general data like, email, fiscal code and some others. Moreover, in the bottom part of the window there are all the categories of data grouped together and the dropdown menu allows the user to move through all the possible information he wants to retrieve. By clicking on the left side menu, in the centre of the page it will appear the group of information asked. Another feature is the dropdown menu in the top-left corner. It is used to filter data with the date in which they were inserted. This is very useful to track diseases and to have some statistics about the patient. By clicking on one date, the application will collect the new data from the database and all the value will be automatically changed. The last feature is the button under the profile picture called “Modifica dati”. If the user tap on that button, he will face a page equals to the right side of *Figure 5.9*. Except to fiscal code, every field will become editable, and the doctor can modify the data of the patient. Once the user has finished to update fields, he can click on the new button in the bottom side of the screen to save the data and to navigate back to the homepage of the application.

5.4.2 Citizen’s profile

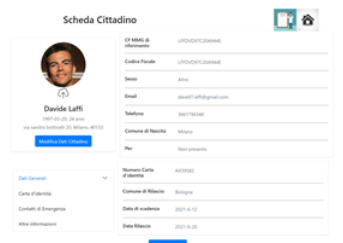


Figure 5.10: Citizen’s profile

The volunteer’s part of the application has a page to visualize citizen’s data too. As it was said before, the user can access it through the list of citizens and selecting a person who has the user as referring volunteer. In the visualization page, the organization of data is similar to the patient’s one. Even for volunteers, there is the possibility, by clicking the button under the profile picture, to make all the fields editable and to change citizen’s data dynamically. In this section it is also present a reset button, which is used to unlink the patient from the respective volunteer. This can be useful if a volunteer change, and so all the related citizens have to find a new volunteer.

5.5 Covid19 emergency

Due to the pandemic situation which involved everyone in the last year, it was implemented a covid related part too. By clicking on covid19 emergency section, the doctor is moved to the list of patients he controls. As before, by clicking on the checkbox he can see the list of all patients saved in the database. The list is exactly the same of the one in *Figure 5.8*. Then the doctor can select either one of his patients or someone else. In the first case he will be able to use all the features of the application's section, otherwise he will not be allowed to add new reports.

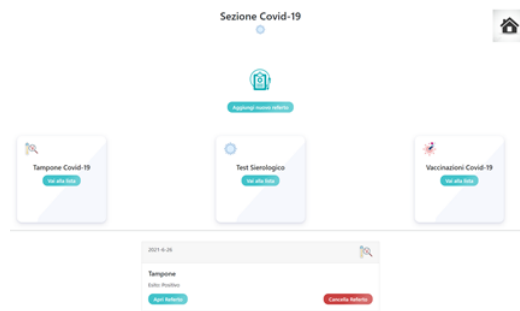


Figure 5.11: Covid19 section

In *Figure 5.11*, it is shown the covid19's section if the patient selected by the doctor is one of his patients. The differences with an external patient is that there is not the possibility to add a new report and to delete a report previously added. This is because reports must be handled from the patient's doctor. As it was said before, there are three typologies of report that can be uploaded to the system. The user can choose among that three categories and see a list of reports uploaded belonging to the specific category under the separating line. For each report there is the date on which the test or the vaccination was performed in the corner, the type of test and the result. By clicking on the light blue button, the application redirects the user to the PDF file of the report that can be downloaded, while the red button is used to delete the report from the database.

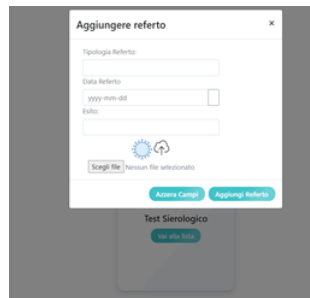


Figure 5.12: Add new report

By tapping on the button add new report, the application shows to the user a small form which asks the information of the report. The first one is a selection list which asks for the typology of the report among the three possibilities. Then there is an input calendar in order to put the data of the test and a result box which contains “Positive” and “Negative” if the report is a test. While, if the typology is a vaccination, the result box is hidden, and there are two more boxes: one for the dose of the vaccine which can be first dose or second dose and another with the type of vaccine given. At the end there is an upload button to save in the database the PDF file containing the report.

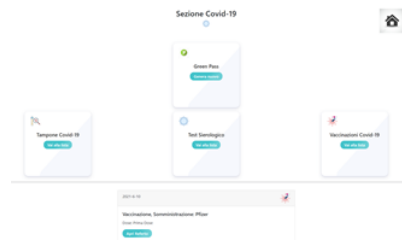


Figure 5.13: Covid19’s section volunteer

Going back to volunteer’s side, the covid19 section is similar to the doctor’s one. Clearly the volunteer is not able to add new report to the system, because it is an exclusive medical task, but he is allowed to see all the previous tests and vaccinations performed by the citizen. Moreover, the volunteer can generate the green pass, which is the European certification codified in a QR code that allows to circulate without restriction in Italy. When the user presses the green pass section, an algorithm computes if the citizen can have it or not. The requirements are:

- **Negative covid19 test in the last 48 hours:** The algorithm checks

if there are negative covid19 tests uploaded for the citizen in the last 48 hours.

- **A full vaccination:** The algorithm checks if there are vaccinations related to the citizen and if the vaccine has two doses and it was done in less than 9 months, the system generates the green pass.

5.6 User's profile

Gestione profilo personale

Davide Laifi
davide.laifi@gmail.com

Nome: Davide
Cognome: Laifi

Telefono: 3661796348

PEC: Non c'è

Salva Profilo

Giorni e orari di lavoro
24/05/2021

Figure 5.14: User's profile

The last section of the application is the user's profile section. Until now, it is the same both for doctors and volunteers, in future improvements the two profiles will have more features and fields and they will be different. On the left side there is the user's name with email and profile picture, with the possibility to change it. Then in centre part of the page there are the general data like name, surname, and phone number and in the right side there are the working days which is one of the future improvements. By clicking on save profile the modifications are saved and data will be registered in the database.

Chapter 6

Conclusions and future improvements

Digital Healthcare tries to link together the part of inserting medical and social data and the part of using them to help rescuers to save citizens' lives. The system is completely implemented and it can be reached at [15]. Now the web application is hosted on Firebase which provides a way to deploy both web applications and mobile ones. The data used in the web application are mock data used only for demonstration. The goal of Digital Healthcare is not to apply as an outstanding and innovative system to collect citizens' data and use them in the best way as possible. The aim is to show that medical information and general social information can collaborate to help in first medical aids. The first chapter tells us some limits and issues of the state of the art both in Lombardia region and in Italy, and Digital Healthcare tries to propose a way of solving the heterogeneity of data and a way to collect them with a very intuitive system. The power of Digital healthcare is that these information are immediately used to provide help in first medical aid, and this is possible because the web application used to collect data, and the mobile application that uses them communicate with a common database and are part of the same system. As it was said before the applications are running and can be reached surfing on internet, but only with a demonstration purpose. To become a system that is used by real doctors and municipal volunteers, there should be access to personal medical information and other authorities come in. As a matter of fact, these privacy problems go over the thesis' purpose and work. Moreover, the Covid19's section was a very interesting improvement. Due to the current historical circumstance, we thought that a medical application requires a section relative to the pandemic situation. In the web application, the doctor is able to add covid report related to patients, while the municipal volunteer

can visualize report and print the Green Pass. This is very useful for elder people which are not able to download a mobile application and to log in in order to download a QR code, but they usually go to municipal hall to asks for documents. The system supports only covid tests, vaccinations, and green passes, but the pandemic situation is constantly evolving and the application will require updates to be still efficient. The hope is that our work can inspire someone, showing that in the current situation there are too many ways of collecting the same data and linking everything together with a unique system can provide benefits in terms of efficiency, but also in first medical aids.

Bibliography

- [1] Fascicolo Sanitario Lombardia
<https://www.fascicolosanitario.regione.lombardia.it/>
- [2] DPCM 31-03-2014 FSE guidelines
https://www.agid.gov.it/sites/default/files/repository_files/linee_guida/2_fse_linee_guida_dpcm_31032014.pdf
- [3] Medici Volontari Italiani
<https://www.medicivolontaritaliani.org/>
- [4] Comune di Milano
<https://www.comune.milano.it/aree-tematiche/servizi-sociali/raccolta-dati-personali-per-interventi-di-emergenza>
- [5] JotForm
<https://eu.jotform.com/>
- [6] Angular Framework
<https://angular.io/docs>
- [7] Firebase
<https://firebase.google.com/docs>
- [8] Typescript
<https://www.typescriptlang.org/docs/>
- [9] Flutter SDK
<https://flutter.dev/>
- [10] Angular Tutorial for Beginners: Learn Angular and TypeScript
<https://www.youtube.com/watch?v=k5E2AVpwsko&t=6239s>
- [11] Codevolution
https://www.youtube.com/channel/UC80PWRj_ZU8Zu0HSMNVwKWw

- [12] Build a CRUD App with Angular and Firebase
<https://developer.okta.com/blog/2019/02/28/build-crud-app-with-angular-and-firebase>
- [13] Angular File Upload - StackOverflow
<https://stackoverflow.com/questions/47936183/angular-file-upload>
- [14] Router Guards
<https://codecraft.tv/courses/angular/routing/router-guards/>
- [15] Digital Healthcare web application
<https://digital-healthcare-it.web.app/>